**Assignment: Employee Management System**

**Objective**:

Develop two Spring Boot applications, App1 and App2, to manage employee data. App2 will handle the database operations, while App1 will expose REST endpoints to receive requests from external consumers. App1 will connect to App2 via REST. In summary, requests go to App1's REST APIs, which internally call App2. App2 will use Spring Data JPA to store data in an H2 database and send back the response.

**Requirements**:

**App2: Employee Database**

1. **Database**:
   * Use H2 database.
   * Design a schema with 3 tables:
     + Employee (fields: id, firstname, surname, roleid)
     + Project (fields: id, name, employee\_id)
     + Role (fields: id, name)
   * Establish relationships:
     + Many-to-One between Employee and Project.
     + One-to-Many between Role and Employee.
   * Add indexing for performance optimization with normalization rules.
2. **Spring Boot Configuration:**
   * Use Spring Data JPA for database interactions.
3. **Business Rules:**
   * Create a stored procedure to delete a role in such a way that all employees associated with that role are also deleted, and any projects assigned to those employees are reassigned to a default employee. Call this stored procedure from the delete operation.

**App1: Employee Service**

1. **REST Endpoints:**
   * Expose endpoints for end consumers (e.g., get employee details, add new employee, update employee information). Please see below API specifications.
   * **Implement request validations in App1:**
     + Identify the roles of the employee from request headers.
     + Validate that the role is not null and has a valid length (e.g., between 3 and 50 characters).
     + Ensure the role matches predefined roles (e.g., ADMIN, USER, MANAGER).
     + Return bad request if there is any invalid input for all the request.
2. **Role-Based Access Control:**
   * Implement role-based access control in App1 to ensure that only users with the "ADMIN" role can create or delete employees, while users with the "USER" role can only view and update employee details.
3. **Data Transformation and Validation:**
   * Handle data transformation and validation when creating a new employee in App1, ensuring that the role provided in the request header is valid and transforming the request object to match App2's expected format.
   * Use a mapping framework (e.g., MapStruct) to transform the source DTO to the target DTO.
     + App1 Request Object:

Fields: first\_name, surname, role (in header)

* + - App2 Request Object:

Fields: name, role\_id

1. **Retry Mechanism:**
   * Make your App2 down and see if it retries 3 times.
   * Make your App2 down and when it finishes retrying for the 2nd time, restart your App1.
   * After App1 restarts, make sure the 3rd retry is performed.

**Deliverables**:

1. Source code for App1 and App2.
2. Use maven and java 17.
3. Use Spring Security to secure the REST APIs.
4. Dockerfiles for both applications.
5. SQL script for database schema and stored procedure.
6. Documentation (README) including setup instructions, API documentation (Swagger), and usage examples.
7. Write JUnit tests with more than 90% code coverage.

**REST Operations**

**App1:**

1. **Create Employee**
   * Endpoint: POST /employees
   * **Request Body:**

{

"first\_name": "John",

"surname": "Doe"

}

**Request Header:**

Role: ADMIN

**Response**:

{

"id": 1,

"first\_name": "John",

"surname": "Doe",

"role\_id": 1

}

1. **Get Employee by ID**
   * Endpoint: GET /employees/{id}
   * **Response:**

{

"id": 1,

"first\_name": "John",

"surname": "Doe",

"role\_id": 1

}

1. **Update Employee**

Endpoint: PUT /employees/{id}

**Request Body:**

{

"first\_name": "Jane",

"surname": "Doe"

}

**Request Header:**

Role: USER

**Response:**

{

"id": 1,

"first\_name": "Jane",

"surname": "Doe",

"role\_id": 2

}

1. **Delete Employee**
   * Endpoint: DELETE /employees/{id}
   * **Response**: json { "message": "Employee deleted successfully" }

**App2:**

1. **Create Employee**
   * Endpoint: POST /api/employees
   * **Request Body**:

{

"name": "John Doe",

"role\_id": 1

}

**Response:**

{

"id": 1,

"name": "John Doe",

"role\_id": 1

}

1. **Get Employee by ID**
   * Endpoint: GET /api/employees/{id}
   * **Response:**

{

"id": 1,

"name": "John Doe",

"role\_id": 1

}

1. **Update Employee**

Endpoint: PUT /api/employees/{id}

**Request Body:**

{

"name": "Jane Doe",

"role\_id": 2

}

**Response:**

{

"id": 1,

"name": "Jane Doe",

"role\_id": 2

}

1. **Delete Employee**
   * Endpoint: DELETE /api/employees/{id}
   * **Response**: json { "message": "Employee deleted successfully" }